RADC-TR-90-276
Final Technical Report
November 1990

ADAPTIVE INTERFACES

Lockheed AI Center

Sherman W. Tyler

Rome Air Development Center
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Services (NTIS) At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-90-276 has been reviewed and is approved for publication.

APPROVED: Sharon M. Walter

SHARON M. WALTER
Project Engineer

APPROVED: Raymond P. Urtz Jr.

RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER: B G Oaks

BILLY G. OAKS
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COES) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	November 1990	Final Mar 89 - May 90

4. TITLE AND SUBTITLE
ADAPTIVE INTERFACES

5. FUNDING NUMBERS
C - F30602-87-D-0087
PE - 62702F
PR - 5581
TA - Q4
WU - C1

6. AUTHOR(S)
Sherman W. Tyler

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Lockheed AI Center
3251 Hanover Street
Palo Alto CA 94303

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Air Development Center (COES)
Griffiss AFB NY 13441-5700

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RADC-TR-90-276

11. SUPPLEMENTARY NOTES
RADC Project Engineer: Sharon M. Walter/COES/(315) 330-3577

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The CHORIS software system is an intelligent interface architecture which supports mixed modality input and output, high-level plan management, adaptation to individual users and user roles, and tailoring of system response information to the currently executing high-level task and the user's preferences. CHORIS has been extended to support users with different roles within the command-and-control-style domain of emergency management. The interface demonstrates the powerful effects that the functionality captured within CHORIS can have in enhancing user performance in responding to complex situations in a timely manner.

14. SUBJECT TERMS
Interface, Command and Control, Adaptive Interface, Intelligent Interface, User Modeling

15. NUMBER OF PAGES
36

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

Adaptive Interfaces
Final Report
Contract No.: PAR-SC-87-0087-06
Principal Investigator: Sherman W. Tyler
Associate Investigators:
Tim Bickmore, Linda Cook, John Tortorice, Robert Gargan
April 1990
Lockheed Missiles & Space Company, Inc.
Lockheed Artificial Intelligence Center

1 Purpose of Contract

The purpose of this research and development endeavor was to design and implement an adaptive intelligent interface for a command-and-control-style domain. The primary functionality of the resulting interface was to be able to adapt its presentation of information to the individual user, based upon the current task and the user's personal preferences and experiences as captured in an explicit user model.

2 Domain

To fulfill this purpose, the domain of emergency crisis management was chosen. This is clearly a type of command-and-control domain, and one that is complex enough to exercise the issues of concern for this effort. In this application, the user views a map of a geographic area and must respond appropriately when some emergency situation arises. If, for example, there is a hazardous material spill, the user sees map information reflecting that spill and might have to determine such things as where to set up the team of people to contain the emergency, whether the area must be evacuated and if so, in what way and to what new location, and so on. A range of different users might be using the system, from someone in charge of state-wide efforts to a person in command at the actual emergency site. A variety of emergency situations might also occur, including earthquakes, floods, fires, and airplane accidents.

The new domain is a type of geographic information system (GIS). A GIS is a computerized database management system for capturing, storing, retrieving, analyzing, and displaying map data. Its notable features are an ability to overlay different sets of data onto a single map and to do multiscale map displays. These systems are in wide use for such activities as managing natural resources, analyzing census data, and assessing natural hazards. In Lockheed, large efforts are underway in both C3I and anti-submarine warfare to apply GISs.

3 Tasks

The work encompasses 5 main technical tasks, as follows:

1. Analysis of Current Interface Technologies
2. Delineation of User Roles
3. Development of User Models
4. Design of Interface Architecture
5. Development of Feasibility Demonstration

The sections that follow will give a summary of how each of these tasks was accomplished and the results of that effort.

4 Analysis of Current Interface Technologies

A wide range of devices and modalities for possible use in this domain was considered. This analysis concluded that the technologies of most value for emergency management were these. For the input of commands and requests, menu selection, direct manipulation, and natural language, using the combination of a mouse and keyboard, were found to be most useful. Menu selection permits easy selection and execution of commands and clearly indicates the range of actions available. Direct manipulation is appropriate because users in this domain often need to interact directly with objects shown on the map, to ask for information about those objects (e.g., the bedcount of a hospital) or to indicate that the object is a parameter for a command. Natural language enables users to ask for information by describing what is needed when direct manipulation cannot be easily used. Thus, for example, a user might want to know "What hospitals in San Mateo County have a bedcount of more than 100", and this could most easily be accomplished through natural language. For output of information, the best options involve the use of high-resolution color displays to present information in the form of business graphs, changes to the map display, or simple natural language sentences. In certain settings, the use of speech input and output might be helpful as well, but given the current state of the art and the frequently noisy environment of an emergency environment, its incorporation in an interface for use during an actual emergency might be premature.

5 Delineation of User Roles and Development of User Models

Information needed for the purpose of modeling the important elements of this domain was obtained from a variety of government agencies. Especially helpful were individuals working in the California Office of Emergency Services (OES), both at the state

capital and at one of the more progressive regional centers (San Mateo County). These individuals provided both detailed verbal information as well as books summarizing general emergency management procedures and analyses of responses to previous actual disasters. Using these various sources of information, models were developed for the domain objects, commands, tasks, and users. The resulting user models are now described. In a later section, the other models will be discussed.

User models for the new domain had to be constructed. In an emergency situation, many different individuals may have some role to play. This can range from those trying to contain the local emergency at what is called the Incident Command System (ICS), to the person in charge of the entire affected region (such as a county EOC administrator), to the Federal government personnel at the Federal Emergency Management Association (FEMA). In the initial version of the prototype, two distinct user models were designed.

One type of user modeled by the system is the ICS Commander. This is the person in charge of the team trying to contain the immediate emergency. For example, in a hazardous material spill, this would be the person directing the group trying to contain the spill and protect those in its immediate vicinity. This user is concerned with setting up the local communications network, viewing the local road system, and warning individuals in the area about the disaster and possibly pick-up points for evacuation. Figure 1 shows a sampling of the kinds of information stored in the current implementation of this user model for the CHORIS interface. All knowledge bases in CHORIS are represented as CommonLisp Object System (CLOS) objects. As Figure 1 shows, the ICS Commander user model includes such object slots or attributes as the commands preferred by this type of user, the tasks typically executed, the commands and tasks this user is not permitted to carry out, and information on how the screen display should initially be organized for this user role.

The second user model embodies the Emergency Operations Center (EOC) Commander, the person in charge of handling the emergency at the regional level. This person must deal with all emergency incidents in the entire region (typically a county), deciding how to allocate available regional resources to each. This user would activate the ICS initially when the emergency occurs, would be concerned only with the main regional highways, would try *what if* questions covering the entire county, and in situations such as evacuations, would be concerned with the destination of the evacuation and getting persons from the incident area to the destination. Figure 2 shows a sampling of the typical attributes and attribute values for this user type.

6 Design of Interface Architecture

This work was based on extending the CHORIS (Computer-Human Object-oriented Reasoning Interface System) architecture developed by the Intelligent Interfaces Project at the Lockheed AI Center. The CHORIS system is a knowledge-based intelligent interface. Its knowledge bases include models of the user and domain, descriptions of domain

```
(defclass [illegible] (choris-UserModel)
   (:documentation "This is the user model for the EOC Commander")
   ((CmdPrefs  :documentation "list of commands user prefers"
     :initform '(set-map-center set-map-scale show-city-halls
                                show-hospitals show-roads
                                display-icons-by-city
                                display-icons-by-county
                                display-incident-summary
                                clear-icons))
   (AdditionalCmds :documentation "additional commands on persistent cmd menu"
                   :initform nil)
   (TaskPrefs :documentation "list of tasks user prefers"
              :initform '(hazmat-spill
                          earthquake
                          fire
                          airplane-crash
                          train-crash
                          medical-incident))
   (ExcludedCmds :documentation "commands user is not permitted to execute"
                 :initform '(activate-communication-network
                             set-assembly-points
                             activate-hazmat-team
                             ...
                             ))
   (ExcludedTasks :documentation "list of tasks user is not allowed to execute"
                  :initform '(local-situation-gaming))
   (InitialWindows :documentation
                   "List of windows that will appear on screen when user
first logs on"
                   :initform '(map-window map-commands agenda-cmds
                                          event-logger
                                          nl-query netcom icons))
   (AllWindows :documentation
                   "Total list of windows - note that some will not be
exposed at the start"
                   :initform '(map-window map-commands nl-query
                                          context-graph
                                          event-logger netcom clipboard
                                          ...))
   (LayoutPreferences :documentation
                      "The layout preferences for the EOC Commander
                           as needed by the Display Manager"
                      :initform (pcl::*make-instance
                                 'interfaceview
                                 :map-window
                                 '(:region (0 300 500 599)
                                           bay-area
                                           ...
            ))
  (:metaclass choris-metaclass))
```

Figure 1: Sample of the EOC Commander User Model.

```
(defclass ICS-Commander (CanonicalUserModel)
  ((:documentation "This is the user model for the ICS Commander")
   (CmdPrefs :documentation "list of commands user prefers"
             :initform '(display-incident-summary clear-icons set-map-center
                                                  set-map-scale
                                                  display-icons-by-city
                                                  display-icons-by-county
                                                  hardcopy-summary))
   (TaskPrefs (:documentation "list of tasks user prefers"
              :initform '(hazmat-spill
                          earthquake
                          fire
                          airplane-crash
                          train-crash
                          medical-incident))
   (AdditionalCmds :documentation "additional commands on persistent cmd menu"
                   :initform '(locate-hospitals show-roads
                                        activate-communication-network))
   (ExcludedCmds :documentation "commands user is not permitted to execute"
                 :initform '(notification
                             get-icon-info-by-distance-from-point
                             number-of-people-affected
                             ...))
   (ExcludedTasks :documentation "list of tasks user is not allowed to execute"
                  :initform '(eoc-loc-determination
                              regional-situation-gaming))
   (InitialWindows :documentation "List of windows that will appear on screen
                   when user first logs on"
                   :initform '(map-window map-commands cmd-task-graph
                   ...))
   (AllWindows :documentation "Total list of windows - note that some will not
                   be exposed at the start"
                   :initform '(map-window map-commands context-graph
                                          event-logger netcom clipboard
                               ...))
   (LayoutPreferences :documentation "interface view for ICS COMMANDER"
                      :initform (pcl::*make-instance
                                 'interfaceview
                                 :map-window
                                 '(:region (0 300 500 599)
                                           bay-area
                                           (:min-longitude -122.850716
                                           ...)))
  (:metaclass choris-metaclass))
```

Figure 2: Sample of the ICS Commander User Model.

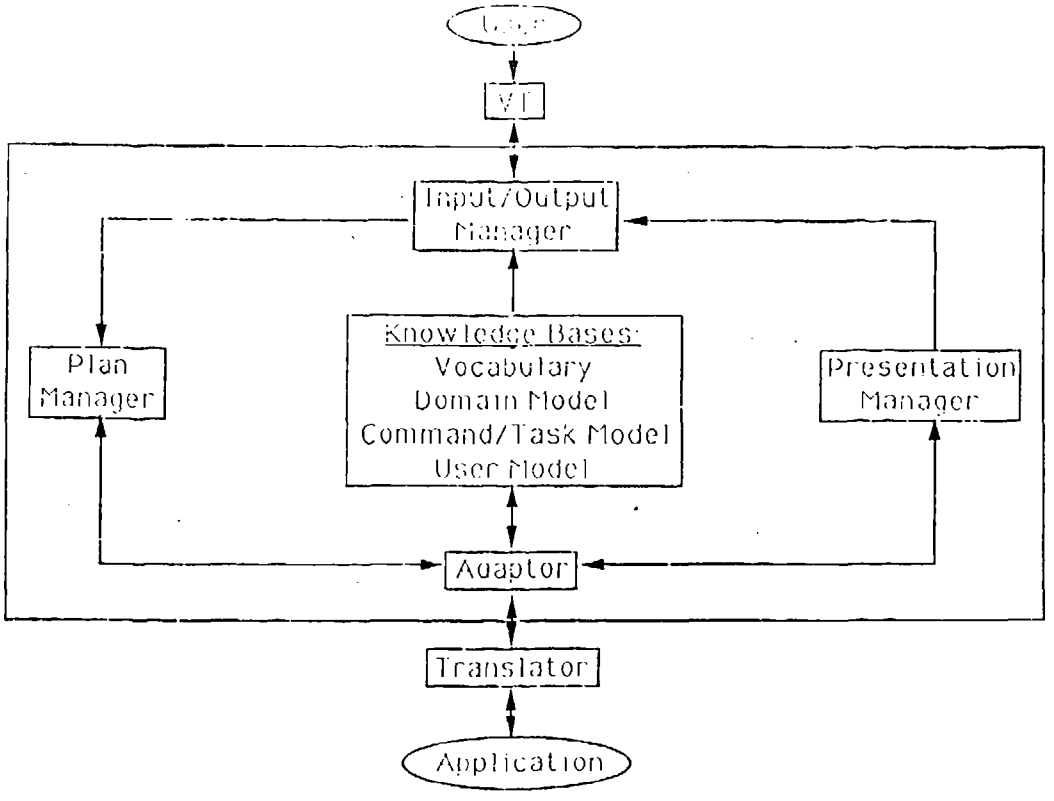

Figure 3: The CHORIS Architecture.

commands and tasks, and a representation of the domain vocabulary. The nature of each of these knowledge bases in the emergency management domain is discussed more fully later. These knowledge bases are in turn used by several domain-independent reasoning modules (see Figure 3). These reasoning modules are briefly described next, followed by a more detailed treatment of each major reasoning component.

The Virtual Terminal (VT) module contains those aspects of the interface that are tied to the particular hardware the interface is running on. The Input/Output Manager is where the input from the user is integrated and converted from its human understandable representation to the logical form used by the interface for reasoning purposes (and vice versa for output); this module also executes display commands. The Plan Manager analyzes user input and produces a representation of the high-level intention(s) of the user, which becomes the basis for developing a plan to accomplish the implied goals of that user on the target application. The Adaptor module is responsible for monitoring the actions of the user and adapting the features and behavior of the interface to best suit that user and the currently executing task. Finally, the Presentation Manager takes the information returned from the application as the consequence of a user request and dynamically determines how to best present this information. A more detailed treatment of these reasoning modules now follows.

7 Input/Output Manager

The primary purpose of the Input/Output Manager is to integrate multimodal input (e.g., mouse picks and natural language) into a single logical representation that can be used by the interface to reason about the user's actions. On the output side, it takes the high level commands from the Presentation Manager and translates them into the lower level generic operations required by the Virtual Terminal. A major focus has been on integrating the natural language (NL) processor into the overall architecture, having it serve as a useable tool for obtaining information and for taking actions. This component aims at making it easier for users to communicate intent. Supporting NL and mixed modes of input allows for naturalness of interaction; the user is free to use the direct manipulation or NL descriptions, whichever is easier.

This work built on the natural language processor *Chat-80* developed by Fernando Pereira. Chat-80 is a natural-language query-answering system implemented in Quintus Prolog. The research done in modifying Chat-80 was done in collaboration with Fernando Pereira and Phil Cohen of the AI Lab at SRI International. The modifications to Chat-80 consisted of adding the capability to 1) interpret deictics (pointing acts as with the mouse to elements on the screen) that are inserted directly into the lexical string being entered by the user; 2) process imperative sentence structures to support NL annotation of commands using forms; and 3) maintain a graphical representation of the discourse context for anaphor resolution. This section presents a brief examination of this component (for detailed information, see [Cohen89]).

Deixis — In everyday language, deictic utterances, consisting of words such as *this, that,* and *those*, typically accompanied by a pointing gesture, are often employed to point out something directly. Deictics can reduce referential ambiguity and enable succinct NL input by increasing the communication bandwidth. In the CHORIS interface, a preprocessor handles the insertion of deictics into the natural language input window.

An example of the use of deictic reference in the emergency management domain would be the user typing "What is the capacity of these schools *pick pick pick* ?". The three "picks" would be mouse selections from the graphical depiction of the schools on the map and would produce a query with the systems's names for each *pick* entered in the NL string. Equivalently, the user could have typed the full name of each of the schools.

As can be seen from the example, the use of deictics provides an elegant way to combine the generality of NL with the specificity inherent in the graphic items depicted on the computer screen to provide the user a more natural method of interaction.

NL Annotated Commands — A common feature of user interfaces developed in the last few years is to have mouse sensitive "menu" commands (e.g., **DELETE, MOVE, OPEN**) that, when selected, activate a prescribed action. Often these commands require the entry of an argument, either by selecting a mouse sensitive item on the screen or by typing in the argument (e.g., the **OPEN** command would require the entry of a file name as the argument.) Modifying Chat to handle imperative sentences was done to add considerable flexibility to the ways in which users can specify commands

and the degree of structure provided by the system in support of command entry.

At present three levels of commands have been implemented: 1) the standard method as described above, which is the most structured and least flexible; 2) a forms based version that allows the user to enter NL phrases in the argument slots; this method provides the structure of the standard method, but adds considerable flexibility in specifying the arguments; 3) complete NL entry of command as an imperative sentence; this method has minimal structure but maximal flexibility. The forms mode of command entry provides the user with a fairly high degree of structure. The command options are selectable from a permanent menu or submenus thereby making clear the legal command options and reducing the burden on the user's memory. By adding the option of using NL phrases as the arguments, the user now has flexibility in specifying the command arguments and can easily initiate actions which would be most cumbersome within a direct manipulation style of interface.

Graphical Anaphor Resolution — Anaphora occurs when a speaker uses certain words (anaphors) to *refer back* to people, objects, events, etc. that were previously mentioned in the discourse context. Although anaphora is used effectively in conversations, the heavy reliance on memory can contribute to misunderstanding and confusion when the memories ('discourse models' in linguistic terms) of the listener and speaker do not correspond. When a NL processor is the 'listener' this problem is exacerbated (the maintenance of a good discourse model is computationally very expensive).

In analysing the NL requirements for various domains, many situations were found to benefit from the capability to use anaphoric references. For example, the person responsible for product assurance might be trying to trace down the source of a manufacturing problem. To do this, he might begin asking questions that become increasingly more specific. In these situations the individual is branching down a decision tree in pursuit of a solution to the problem. That is, he is establishing a dynamic discourse model in which each question builds on the previous questions and their answers.

To accomplish this task using a standard database query language would require the user to build longer and longer queries as the questions become more and more specific. A more parsimonious approach is one that permits the user to ask simple NL queries that refer to elements of the discourse context established by previous questions.

To avoid the computational problems associated with currently offered processing solutions to discourse models, a graphics based method for explicitly maintaining and presenting a discourse model was developed. Figure 4 shows an example of a window that is generated in response to a NL query. These windows are the basic elements of the graphical discourse model. Each window has attached to it Contexts that allow the user to ask follow up questions referring to just items that are in a particular context. This restricts Chat's query planner to query the database only about those items that are in the current context. In the example window the context is TEST. To ask a question just about the tests shown in the example window, the user would select TEST and then type the question after the prompt "FOLLOW UP:" in the figure. The system's response will be just for those tests.

A question history window displays the interrelationship among all the questions

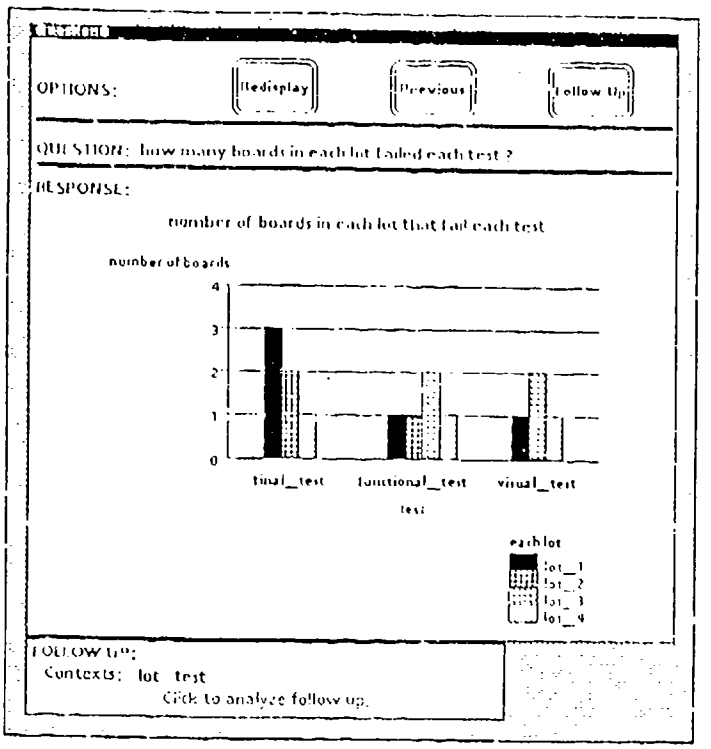

Figure 4: An Example Response Window.

that have been asked. It can be called up as a system facility by opening the "?" icon. If the user needs to ask another question in relation to a previous one, for example, s/he can pick that response in the Question History Window and the system will redisplay that window. The user can then enter a new follow up question in the contexts available at that point. The system will display the response, and the question will be added to the question history window (see below for more discussion on this facility).

CHORIS provides the user with the capability to make anaphoric references to previously answered questions by combining a graphic context environment with NL. Because the system handles the context the user no longer has to build more and more specific queries. The graphics interface makes clear what the current context is and the series of questions that led up to the current response.

8 Plan Manager

The Plan Manager has as its primary function assisting the user through knowledge about typical plans for achieving high-level goals on the target application. Operating from knowledge of the user's current goals and plans, the interface can: detect and try to correct global errors, errors in user plans that would not otherwise be regarded as mistakes; complete high-level tasks, as by filling in default parameters and directly executing the system commands composing the plan steps; interpret ambiguous requests; indicate the current state of an executing plan; and in general help the user in mapping

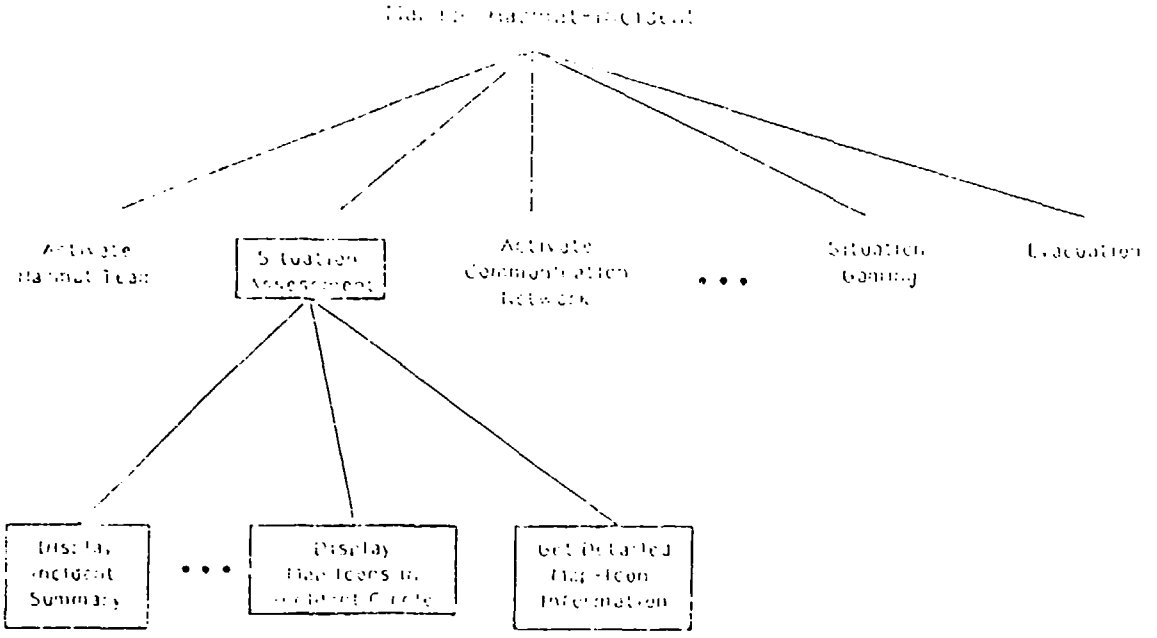

Figure 5: An Example Task Hierarchy: Manage Hazmat Incident.

high-level goals into the low-level commands of the application.

Such a plan-based approach requires three critical elements: 1) a declarative representation of plans; 2) routines for utilizing such representations to directly assist users in their interactions, as described above; and 3) the reasoning ability within the interface to determine what particular goals users are trying to achieve. To date, substantial progress has been made on the first two elements. A useful way of representing plans as a separate knowledge base within the interface was developed. Plans in the interface are represented as object hierarchies, with each object standing for a substep of the plan. Figure 5, for example, shows the object hierarchy for the *Manage Hazardous Material Spill* task in the emergency management prototype. Thus, when an ICS Commander must manage a hazardous material spill, the system knows that this task consist of such steps as activating the hazmat team, doing a situation assessment, activating a communication network, and so on. Thus, the top of the tree represents the high-level goal itself, non-leaves constitute substeps of the plan for achieving that goal, and, at the lowest level of the hierarchy, the leaves stand for actual commands that can be given to the system.

Each step of the task, as an object, is described by a number of variables or slots. The most critical variable is a list of the immediate substeps of the given step and of the interrelationships among those substeps. The interrelationship information includes the required preceding and following substeps of a given substep, as well as the minimum and maximum number of executions of that step permitted in executing the task.

The second main element of the Plan Manager, namely, the routines to support assisting users based on their current goals, has also been realized to a significant degree.

Thus, the interface can consider the commands and arguments to commands provided by the user, compare those with the constraints on the task substep's parameter values, and thereby detect global errors. This same mechanism allows the interface to make reasonable guesses, about what default parameters are appropriate for the current task step, and to provide these automatically for the user. In this way, ambiguous commands and incomplete commands can be specified without the need for precision and completeness on the user's part. The interface can also provide an indication of the current state of the plan. This is done by graphically displaying the task hierarchy in a separate window on the screen, with the currently executing step highlighted and those steps already executed shown boxed. The current interface prototype supports executing several tasks at once, or suspending the execution of one plan and resuming it at another time.

9 Adaptor

When the range of user abilities and roles is large, or there are varying constraints on task execution (as rapid vs. self-paced response times), having a single interface for all situations is inadequate. Different users will need more or less descriptive information on interface features, will require varying commands in interacting with the system, and will operate best when their view of the domain is tailored to their role and when the format of the information returned from the system supports the kind of decisions they must make. Therefore, another powerful capability for an intelligent interface is the ability to adapt its own features to the needs and preferences of the current user and the user's present task.

There are three key issues that must be addressed in designing and implementing the Adaptor: when to adapt, what to adapt, and how to adapt. The solution to these three issues, as well as a discussion of the major components of the Adaptor, will now be considered.

When to adapt - Dialogue Phases. To answer the first question of when to adapt, the system must have some way of looking at the dialogue that permits detecting changes in the current dialogue state that warrant adaptation of the dialogue as the next state is entered. In the Adaptor, this issue can be resolved by viewing the interaction of the user and the computer system as consisting of a series of interaction events, with each event in turn composed of a set of distinct dialogue phases (see [Benbasat84]). An interaction event is an atomic unit that brackets the actions from when a user enters a request or command until the system returns a response. In the basic sequence of phases, the user enters some input, such as a natural language request (InputNLPhase), the response is sent on to the system for execution (ExecutionPhase), and the response of the system is displayed to the user (SystemResponsePhase). Looking at the dialogue as a series of interaction events, each composed of such phases, provides a way of organizing the Adaptor so that: 1) it is triggered at the appropriate times, i.e., when an event causes the dialogue to enter a new phase; and 2) the adaptive mechanism can be applied in a discrete fashion, considering only what can be adapted in the interface for the

impending dialogue phase. This leads to the second issue for the Adaptor.

What to Adapt - Dimensions of Interface Variability. To determine what to adapt, the interface needs to know what the possible range of options is for each major attribute of the interface at any given point in the dialogue, and must be able to select any appropriate combination of those features for dynamic realization. Much of the power of the Adaptor arises from its interaction with the other interface modules: calling upon the user model to interpret ambiguous input, reduce the search space of possible user intentions, and indicate how best to organize the interface to meet user preferences. In isolation, the Adaptor in the emergency management prototype of the interface currently considers four primary dimensions of variability. The first is the screen view of the geographic area, that is, the map. The system can present various geographic areas and can scale these to varying levels of detail, depending on the needs of the user. The interface is also able to vary the commands provided to the user for direct or forms-based execution, and can similarly vary the high-level tasks made available in the agenda or task hierarchy windows. An EOC Commander, for example, could be directly supported in taking the necessary steps to handle an earthquake incident. The interface can also select certain facilities, such as the Event Logger or the Netcom (see below), to be open by default for a given user. Finally, the interface can vary the way in which data is presented to the user, showing users information in their preferred mode of viewing.

How to adapt - Main components. Figure 6 shows the major components of the Adaptor and how they interrelate. Each of these components is instantiated in the interface as a separate object or group of objects (in software terms). Aside from the knowledge bases, the main components of the Adaptor are these: the Monitor, which updates each of the different knowledge bases as conditions change; the Accessor, which provides generic functions for obtaining information from each of the knowledge bases, allowing other interface components to use them without having to know the internal details of their structure; the dialogue phases, each of which has parameters describing how the interface should look when that phase is entered as well as rule sets for adapting the interface for that phase; and an ExecutiveRouter, which decides, when an interface event occurs, what the new dialogue phase should be, and coordinates the overall operation of the other Adaptor components.

How to adapt - Operational details. In general terms, adaptation occurs as follows (see Figure 6). When a significant interface event occurs, the Executive Router is called. Based upon the current dialogue phase and the nature of the event, this component determines what the next dialogue phase should be. The original dialogue phase is then called upon. It typically updates some of the knowledge bases, recording in the user model, for example, that a given command was successfully executed from a form by the user. The old phase also adjusts the interface if necessary, as by removing interface features associated with that phase (e.g., a particular menu or form). The Executive Router then calls upon the new dialogue phase. The new phase exercises the most fundamental mechanism of adaptation by firing its particular set of rules. These rules consult the state of the appropriate knowledge bases through the Accessor, and

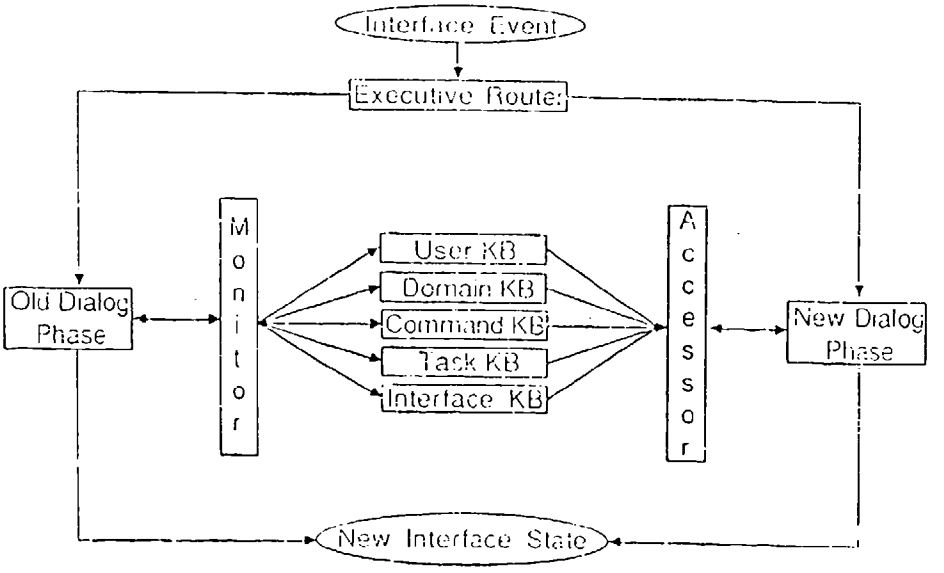

Figure 6: The Main Adaptor Components.

based on the contents of these knowledge bases, the main parameters of that dialogue phase are set (details on the form of the rule sets for each dialogue phase transition can be found in [Tyler86,Tyler88]. The new dialogue phase then determines what features should be present in the interface based on the values of its parameters, and the interface is accordingly modified. The resulting action can range from something very simple, such as constructing a form for filling in the arguments to a command, to something much more complex, such as defining the context for a high-level task, which involves displaying graphically the task substeps and their states (e.g., executed or still to be accomplished), inferring default parameters and actions, dynamically designing menus of substeps, and interpreting user actions in terms of the selected task.

The impact of this module can be seen in a number of the features of the example screen of Figure 7. First, because the user is concerned with emergencies throughout a given region, the view provided by the map (left window) shows the entire Bay Area at a relatively low level of detail. Furthermore, the set of map commands offered in the menu attached to the map have been filtered to include only those actions this user would likely want to perform (e.g., looking at the locations of city halls as possible EOC locations). The system also determines how to show information to the user based upon that user's needs. Therefore, Figure 4 shows a barchart, facilitating detection of trends. Another type of user or task might suggest instead that actual values were more important than trends, in which case, the interface would display the data as a table instead.

None of the decisions made by the interface are final, however. The user is given the option of making other choices. Hence, under the *Map Commands* menu, there is an entry called *Additional Commands* which allows the user to select a command not on

the immediate menu. If the user selects a particular command from that additional list, then that fact is recorded in the user model. If the command is selected often enough, the interface will automatically add it to the main menu. Similarly, when the system generates a response, as in Figure 4, the user can use the mouse to bring up a menu of possible alternative ways of showing the same data. If one of these is selected, the data is redisplayed in that format, and the user model is updated; enough such selections can lead to a decision by the interface to modify the way it shows such data by default (this is also discussed in the next section). The overriding objective of the adaptor module, then, is to assure that the interface is modified to best suit a given individual's needs, thereby easing the user's burden in accomplishing tasks on the system.

10 Response Planner

The final module of the intelligent interface is the response planner. A response planner is needed for several reasons. First, different users prefer to see the same kinds of information in different fashions. Additionally, there are many different types of modalities and techniques within those modalities which are accessible. A response planner can examine the data returning from the application and determine the most appropriate modality and modality technique(s) for presenting the information. Finally, having a response planner removes the requirement that the designers continually redesign and implement the output side of an application each time system requirements change.

There are three steps to planning a response — 1) expressiveness, 2) effectiveness, and 3) adaptation. These steps are incorporated into the selection of modality as well as one or more techniques within the chosen modality for presenting the information to the user.

Rather than describing modality selection and technique selection (which is discussed more fully in [Jr88]), this paper presents the basic process in terms of the three steps introduced above.

Expressiveness Expressiveness is a measurement of a presentation technique's ability to present the given information [Mackinlay86]. During this test, the Response Planner examines and compares the information to the constraints of the various presentation techniques. For instance, during technique selection, if some numerical information, some symbolic information and some relationship between those two fields exists, then a bar chart can be used to fully express the information.

Effectiveness The effectiveness step compares the expressive techniques and modalities within the context of human perception and understanding. Values used for comparison come from a user model which has inherited the values from the canonical user model. The canonical model contains general perceptual information (size of bars in bar charts, line widths, spacing, frequency, etc.) [Cleveland84,Hochberg86,Tukey86] and information comparing relative values of one technique versus another [Clark86,DeSanctis84]. For example, the bar chart's effectiveness step checks for the number of bars. The

greater the number of bars the more difficult it is to distinguish the relationships between any two bars. This is a result of a greater distance between the first and last bar making it more difficult to distinguish relative height.

Adaptation The last step in the response planning process is to adapt the response to the individual user. Here the techniques which have passed the effectiveness test are ordered based on which technique can most support the user's intention. Finally, the response planner presents the information using the technique within that list that the user prefers. So, for example, one user may prefer to see specific numerical information in a table, while another might prefer a bar chart for that same data.

The response planner considers a variety of techniques and modalities each time the user issues a query or a command to the interface in a similar process. As a r ult, it is unnecessary for an interface designer to sit down with the system in order to .ayout all the possible responses. This means that as the users or the underlying application change, it is not necessary to "redesign and construct" the output of the interface.

11 Development of Feasibility Demonstration

The main focus of this research endeavor was to develop a running prototype implementation of the CHORIS architecture in the new domain of emergency management. Figure 7 shows an example screen from this implemented demonstration. To construct the implemented system involved a number of activities. First, although originally developed on Xerox-1186 computers, it was more appropriate to implement the new domain in a more standard environment so that it could be run on widely available hardware, including Sun computers. To this end, the Interlisp code was translated to CommonLisp and and the CommonLisp Object System (CLOS), and the X-Window system was employed for window management. In addition, since the major action necessary to implement CHORIS in the new domain involved building the new domain-specific knowledge bases, some tools were developed or extended to facilitate this process. In particular, a tool to aid with defining the domain objects and their relations was designed.

The actual implementation of an intelligent interface for the emergency management domain required constructing both domain-specific knowledge bases and a number of special interface features or facilities to aid the emergency manager. These are now detailed.

11.1 Knowledge Bases

Aside from the user models, which have already been described, the new knowledge bases required for the CHORIS architecture included the domain objects and the commands and high-l ;vel tasks for acting upon those objects. These are now briefly described.

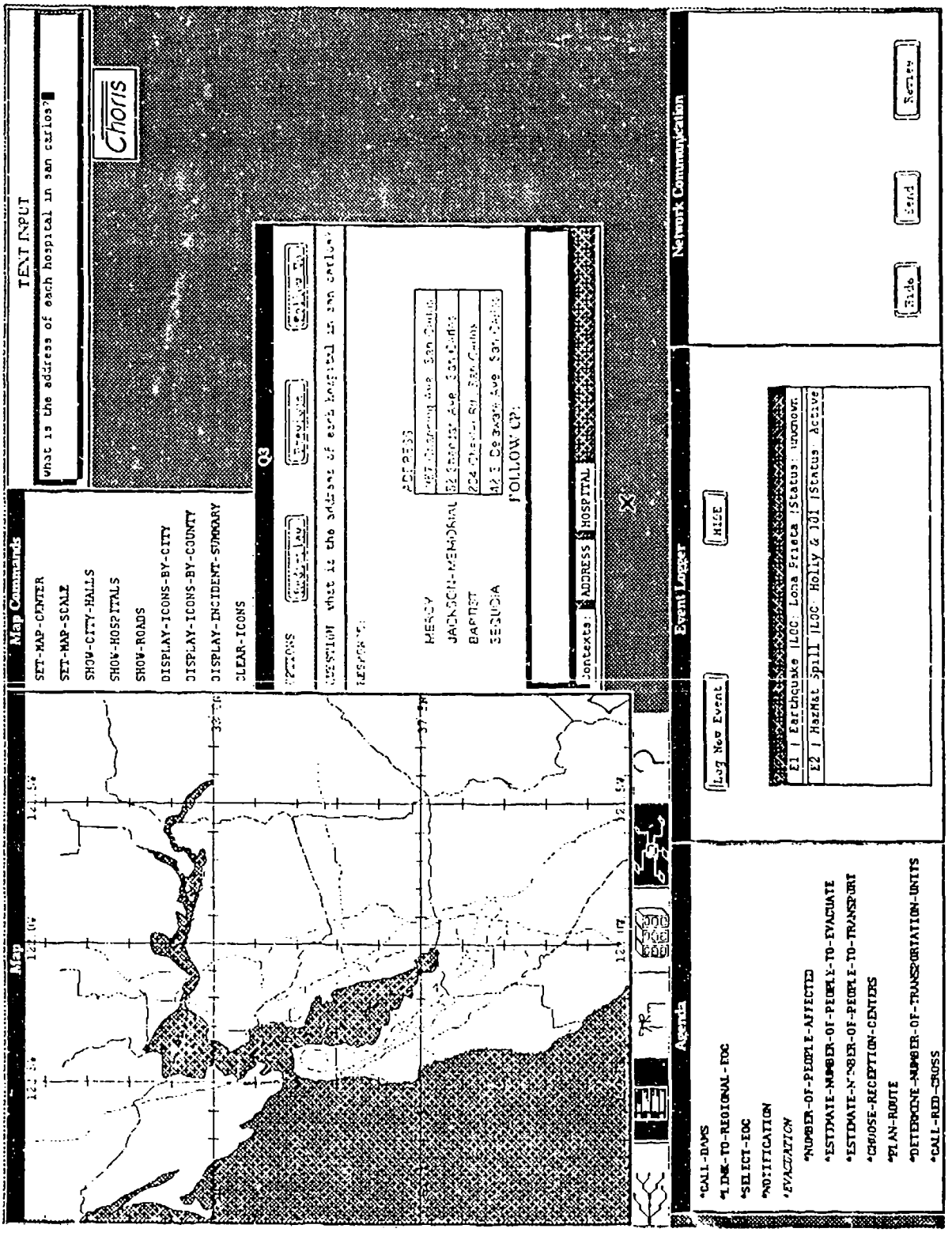

Figure 7: An Example Full Screen: EOC Commander Managing Earthquake

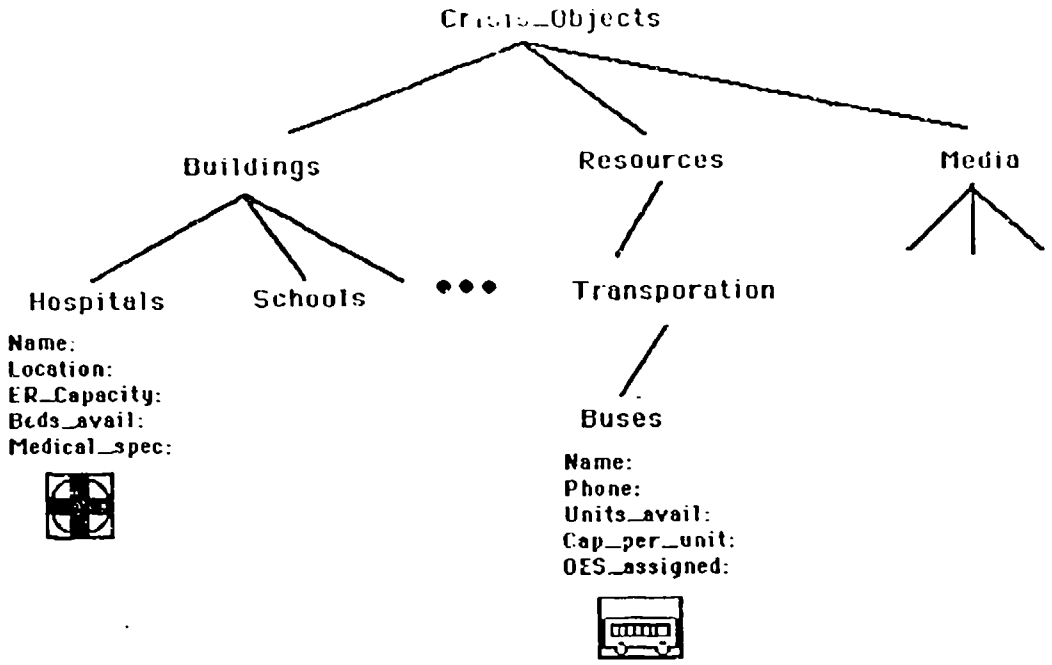

Figure 8: An Example Domain Object Taxonomy.

To capture the important items in an emergency situation, a wide variety of domain objects had to be represented in the CHORIS domain model. A portion of the resulting domain object taxonomy is shown in Figure 8. This included representations for such things as buildings (e.g., hospitals, schools), resources (e.g., transportation), roads, and organizations (e.g., Red Cross, Highway Patrol). Each of these objects was captured as an *object* in terms of a number of key attributes. Especially crucial were the map icons associated with each type of object, since these provided a way of displaying the object visually on the screen and thereby allowing the user to both view and interrogate these objects for crucial information. Figure 9 shows a portion of the representation of a domain object.

The commands and high-level tasks of the domain were also embodied in an object-oriented formalism. Individual commands include such things as displaying an incident summary or printing detailed information on a certain map icon. Many of these commands require some kind of map interaction. The task representation was designed to capture the ways in which experts achieve more complicated goals in the domain through a series of individual commands. An example of a domain task is shown in Figure 5 for managing a hazardous material spill. As shown in the figure, handling such an emergency requires activating a *hazmat*, or hazardous material, team which tries to contain the incident locally; assessing the situation; activating a communications network; doing some situation gaming to try out possible future scenarios; possibly evacuating endangered personnel; and so on. This representation enables the interface not only to guide users in a step-by-step fashion in dealing with emergencies, but also supplies highly useful knowledge for the purpose of training new users. Figure 10 shows part of an example task representation.

```
(defclass buildings (physical-object)
  :documentation "The generic definition of a building"
  ((SystemName :documentation "The system's name for the building"
               :allocation :class)
   (actions :documentation "Actions appropriate to the building"
            :allocation :class)
   (name :documentation "The common name for the building")
   (address :documentation "The street address of the building")
   (people-capacity :documentation "The number of people building holds")
   (adult-furniture :documentation "Whether furnished for adult vs child")
   (general-purpose-room :documentation "Whether large general room present")
   (food-service :documentation "If food available at building")
   (parking-capacity documentation "Number of parking spaces")
   (emergency-pwr-generator :documentation "If has own power generator")
   (site-status :documentation "Status of site - occupied, active, etc.")
   (public-address-sys :documentation "If ability to address over intercom")
   (phone-number :documentation "Phone number of main switch")
   (contact-person :documentation "person to be contacted in emergency")
   (closest-highway :documentation "Name of closest main highway"))
  (:metaclass chris-metaclass))
```

Figure 9: Example Domain Object Representation: Building.

```
(defclass HAZMAT-SPILL (DomainTask)
  (:documentation "Task for HAZMAT-SPILL.")
  ((CallName :documentation "The symbolic name used internally for this task."
             :initform 'HAZMAT-SPILL)
   (OnlyMenu :documentation "The menu entry to display for this command."
             :initform "HAZMAT-SPILL" )
   (OnlyPrompt :documentation "The mouseline prompt for this command."
               :initform "To handle emergency hazardous material spill")
   (PossibleSupersteps :documentation "The possible supertasks of this task."
                       :initform '(MANAGE-CRISIS) )
   (SubstepList :documentation "The subtasks of this task."
                :initform '(ACTIVATE-HAZMAT-TEAM
                            SITUATION-ASSESSMENT ACTIVATE-COMMUNICATION-NETWORK
                            EOC-LOC-DETERMINATION NOTIFICATION
                            Regional-SITUATION-GAMING
                            local-situation-gaming EVACUATION))
   (SubstepReqs :documentation "Parameters of the subtasks of this task."
                :initform
                '((ACTIVATE-HAZMAT-TEAM 1 1 nil
                                        NIL nil
                                        (SITUATION-ASSESSMENT) NIL)
                  (SITUATION-ASSESSMENT 1 99 (ACTIVATE-HAZMAT-TEAM) NIL
                                             (ACTIVATE-HAZMAT-TEAM) NIL NIL)
                  (ACTIVATE-COMMUNICATION-NETWORK 1 99
                                                  (ACTIVATE-HAZMAT-TEAM) nil
                                                  (ACTIVATE-HAZMAT-TEAM
                                                   SITUATION-ASSESSMENT)
                                                  NIL NIL)
                  ...))
   (SubstepRelations :documentation "OR=Any subtask achieves task completion"
                     :initform 'OR)
   (SubstepExecutions :documentation "List of (substep #) pairs giving
                       number of times executed."
                      :initform
                      '((ACTIVATE-HAZMAT-TEAM . 0)
                        (SITUATION-ASSESSMENT . 0)
                        (ACTIVATE-COMMUNICATION-NETWORK . 0)
                        ...))
   (Graph :documentation "graph associated with this task"))
  ...
```

Figure 10: Example Task Representation: Manage Hazmat Incident.

11.2 Map System and Map Commands

In the implemented system, a large portion of the screen is occupied by a geographic map of the region of immediate interest to the emergency manager. The map has two main features: it can be zoomed in or out to show the appropriate level of detail; and it can show and remove a variety of different map objects in an overlay fashion, such as roads, schools, hospitals, etc. The map is designed to convey the current status of the emergency situation to the user, but also to enable direct manipulation kinds of interactions. Thus, if the user buttons on an icon such as a hospital, the system will display information summarizing the details of the object represented by that icon (street location, bedcount, etc.). Furthermore, if the user selects a command that brings up a form and the user buttons on an icon appropriate for the current slot of that form, its relevant representation will be placed in the form's slot.

Associated with the map is a menu of commands that remain permanently available to the user for manipulating the map. This set varies with each user, but can include such operations as changing the map center or scale, showing certain types of objects as icons on the map (e.g., city halls, towtruck companies), clearing selected icons from the map, or drawing the road system. Most commands, when selected, as indicated above, can be filled in through menu selections and simple type-ins or by direct pointing operations to the appropriate map objects. The forms and their slot values are constructed dynamically from the system's command knowledge base, and so the command set and the parameters of any given command can be easily altered by simply modifying that knowledge base.

11.3 The Natural Language Input Window

The Natural Language Input Window, which is located in the top righthand corner of the screen, allows users to type queries to the system in natural language. Typically, in this type of domain, this facility is used to request certain information that cannot easily be asked otherwise. An example sentence might be: "What is the bedcount of each hospital in San Mateo County?" The system passes such queries to the natural language processor, a modified version of Chat'80 which runs in Prolog. The natural language processor returns a logical form representing its parse of the sentence. In this case, the logical form would look something like:
"answer(X) := hospital(Y), in(Y, san mateo), bedcount(Y,X)".
This in turn is used by a translator module to retrieve the relevant information from the knowledge bases.

This natural language capability has most of the same limits in its range of comprehension as the original Chat'80. It can fail for two main reasons: it may not be able to parse the sentence; or it may not be able to find any information in the knowledge bases which satisfies the logical form. The system is currently limited in its feedback to providing users with messages indicating which of these failures occurred. However, there is fairly wide coverage for the types of information emergency managers might need to assist them in reaching their decisions.

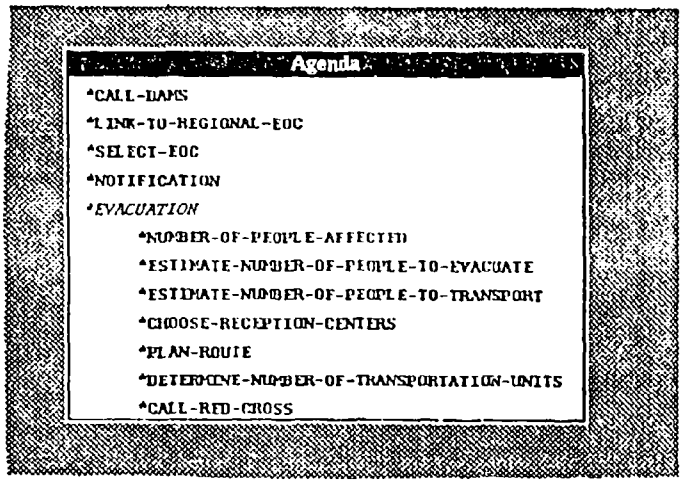

Figure 11: The Agenda Facility.

11.4 The Agenda

One of the main capabilities provided by CHORIS is the agenda facility. The agenda represents a list of commands commonly done by emergency personnel in a similar situation. The commands are organized hierarchically into tasks to help organize the work of the emergency specialist.

To produce the agenda, CHORIS maintains a hierarchical organization of commands and tasks for various application modes. In the current implementation, application modes include Emergency Operating Center (EOC), hazardous material spill, and medical incident. For each of these situations, CHORIS maintains a list of commonly associated commands organized into a task tree. This task tree can be used directly in CHORIS to view the entire structure at once and to execute commands by selecting leaf (command) nodes.

Although this tree can be instructive to view, users in general would find it difficult to navigate the complex task structure. Instead, the tree structure has been collapsed into a scrolling menu facility with indentation to represent a lower level in the tree. Nevertheless, the graphical depiction of the task hierarchy is available for browsing in CHORIS.

CHORIS maintains a significant amount of information associated with each node in the task tree. Tasks are differentiated from commands by having an arrow to their right indicating more information is contained "below" this item. Obligatory commands and tasks are differentiated from optional ones by italics. Further, commands that have been executed are indicated in the agenda as blue. In some cases these items can be executed again, if appropriate.

Initially, CHORIS loads the agenda with the first level commands or tasks for a given situation. Commands can then be directly executed or tasks can be expanded into the next level of nodes. Figure 11 depicts the agenda during an interaction with an EOC officer. In this figure, the "evacuation" task was expanded to reveal the commands within this node.

11.5 The Event Logger

One problem associated with crisis situations is the effective management of an overwhelming volume of information about events that result from an on-going emergency situation. Often, this information arrives piecemeal (reports stagger in, detailing various states of an event), is sketchy or incomplete (i.e. two different reports may describe the same event in a substantially different manner) or is of questionable origin because it comes from undetermined or unofficial sources.

The crisis management unit has a number of responsibilities regarding the management, control and use of this information. One task is to track each event as it unfolds since more information about an event enables crisis managers to make more informed decisions. Thus, it is necessary to support the saving, retrieving and updating of event information as it enters the emergency control unit. Interface support for this necessitates a logging system that is capable of recording important attributes about the incoming event including the time it is logged, a brief message describing the nature of the event, its location, the person reporting the event, to whom the event information was routed for attention and the capacity to update events as new information becomes known. Finally, crisis managers make decisions about events such as the resources that have been allocated in response to that event - and such information should be readily available. A second information processing task that crisis managers require is the ability to look at all on-going events from a number of different perspectives. For example, they may wish to know how many road accidents are currently active and the location of each.

The CHORIS interface has an event logging capability that allows the user to store, retrieve and update event information as it becomes known. It contains a scrollable window which displays all recorded events, in the order that they were reported to the crisis manager. Shown for each event in the window is the event number, the type of event, the event location and, finally, the current event status (either active or closed). Buttoning on each individual event shown in the window allows the user to either display a more detailed record of that event, including all currently known information or to update that event with additional data. The event logger can be activated by selecting the *Log New Event* button. This action results in a new event-report form which the user may fill in with all currently available information regrading the event. That event and its relevant information is immediately visible in the event logger's scrollable window displaying all on-going events. Figure 12 shows an example of the Event Logger during an ongoing emergency.

11.6 The Netcom Facility

The network communication tool (NetCom) is the primary communication link between nodes of the emergency management system. It allows different individuals on different machines to communicate electronically with one another. At login, the user is asked for the role s/he will play in the domain (EOC, ICS) and for the type of incident, and these two facts establish the user's address on the net. Some means for users

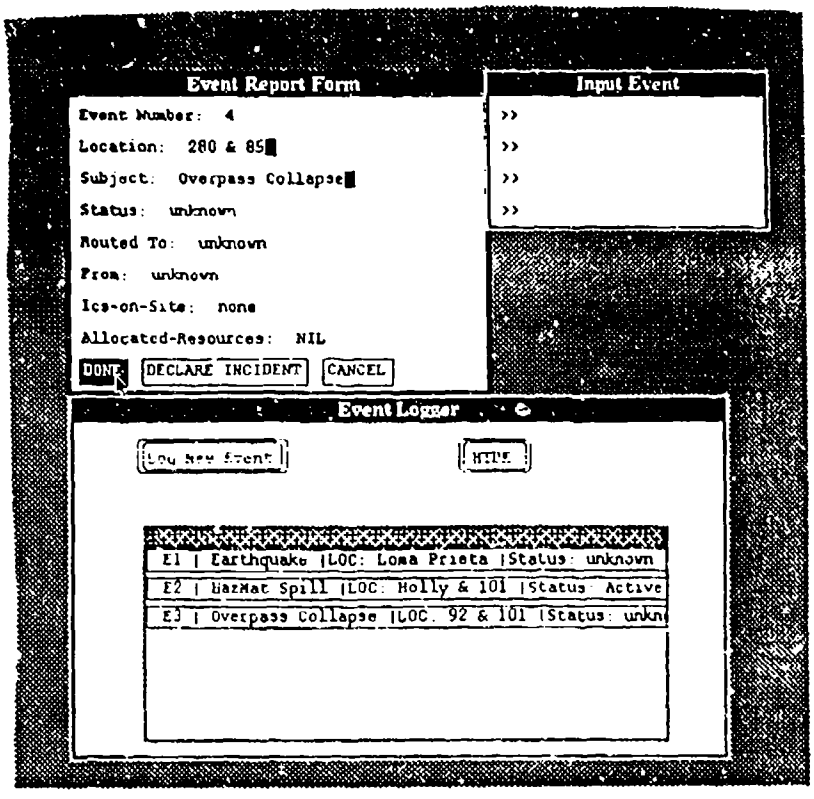

Figure 12: The Event Logger.

to communicate is vital within the emergency management domain. In most current emergency offices, this is done through telephone or walkie-talkies. In CHORIS, it was most reasonable to make it an electronic facility which could function as an integrated part of the entire system.

At tool initialization, an asynchronous process is created which queries the file system for a mail file addressed to the given user. If such a file is found, it is sent to a second process to be handled as an incoming message, and the header of the message is posted to the NetCom window, if it is open, or the NetCom icon is flashed, if it is not open.

The NetCom window itself allows three different actions. The user can mouse the *Send* button to send a message to another node on the network. When this is done, the system presents the user with an input form consisting of recipient, subject, and message sections. The recipient is chosen from a menu of known network nodes (based upon previous logins). The subject line is what will be posted to the recipient's NetCom window when the message is sent. The main text is entered in the message section. Once the subject line of a message has been posted to a given NetCom window, the user can mouse that line to get a menu that allows either deleting it or displaying the message contents in a new window. There is also a button that allows users to review the entire message history (*Review*) and another for hiding the NetCom window. Figure 13 shows an example of the NetCom facility in use.

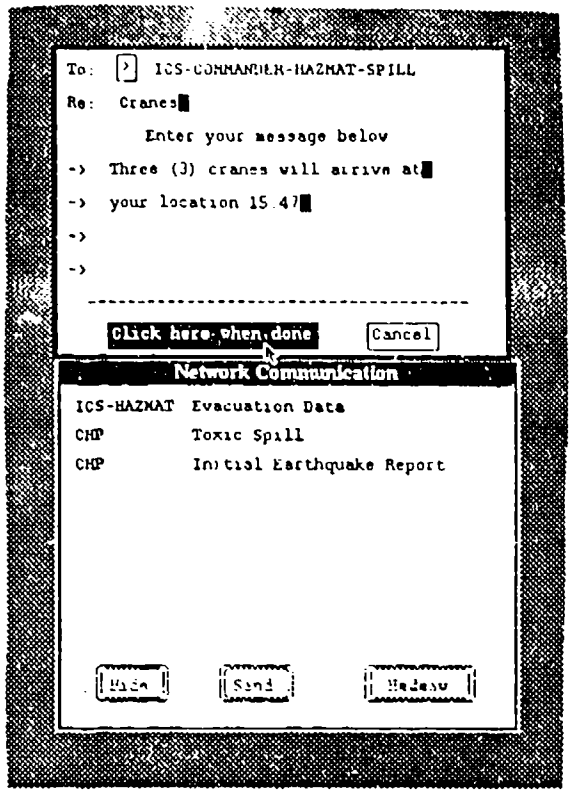

Figure 13: The NetCom Facility.

11.7 The Clipboard

The Clipboard facility is a simple data management window which corresponds in function to the metaphor of a clipboard in the sense of offering a free-form, note-taking capability. Mousing the Clipboard icon opens the window to this asynchronous process. To add information to that window, the user mouses the *Add* button. The response is a new window which provides an area for text input and buttons for signaling the termination of text entry. After the text has been typed and the *Done* button selected, the text is displayed on the main Clipboard window. Text entered in this way can also be removed from the Clipboard by mousing the given text line and selecting *Delete* from the resulting menu. The Clipboard serves the function in the current domain of allowing the user to enter simple reminder messages that may not be covered by the normal task steps as described in the Agenda.

11.8 The Question History

Also within CHORIS, the user has the ability to retrieve answers associated with previously asked questions. These questions are presented in a scrolling menu-style window and selecting a question retrieves its entire response window. Not only is this facility useful for examining responses to past questions, but also the user could issue a follow-up question to the retrieved question. In other words, the user could follow one line of questioning by asking a query and issuing several follow-up questions, decide to

proceed down a separate line of inquiry, and then return to the initial context. This concept of context switching supports, at a higher level, one of CHORIS's key features of direct manipulation of context (see above).

To present the complex hierarchical structure of past contexts, CHORIS uses a flat list that is indented to indicate follow-up questions. This mechanism facilitates navigation through the context tree, which can grow quite large after a sustained interaction. This depiction of a complex hierarchical structure via a flat, indented list is also consistent with another CHORIS facility, the agenda (see above).

11.9 The Resource Allocator

One of the functions of a crisis management team is to keep detailed records of the resources that are available to them in the event those resources are needed in responding to an emergency situation. Examples of resources might include all tow truck companies; alternate modes of available transportation such as buses, trains, ferries; or all hospital facilities. It is also necessary to store with each resource a set of attributes which describe unique characteristics about that resource. As an example, for a resource such as a hospital, it would be important to know what its specialization was (e.g. trauma, burn, or cardiac care) and whether the hospital was accessible by helicopter. For another resource, such as schools (which serve as relocation centers) it would be important to know the capacity of the school's gymnasium or whether there were cafeteria facilities available. However, particularly important for each resource is the availability status. In the event a resource is needed, the crisis manager must be able to determine quickly which resources are available for assignment and which are not.

The Resource Allocation mechanism in CHORIS operates in two different modes: browse and allocate. In the former, the user's intent is simply to view the status of all resources or a single resource. The user buttons on *Browse* and is then prompted by CHORIS to select the desired resource(s). When resources are displayed in the browse mode, there are two accompanying attributes: allocated and available. This is numeric information that indicates to the user the number of units that are currently available should they be needed.

In the allocate mode, the user buttons on *Allocate* and is prompted by CHORIS to identify the type of resource to be allocated and the incident to which the resource is being allocated. When resources are allocated, the total number of units being allocated are subtracted from the currently available units, so the resource knowledge base always reflects any change in the status of a resource that the crisis manager may make. This type of support allows the user instant access to the state of the resource knowledge base. Figure 14 shows a snapshot of the Resource Allocator in use.

11.10 Response Windows

Finally, whenever the system needs to display information in response to a user query and this cannot be done through changes in the map, a response window is produced.

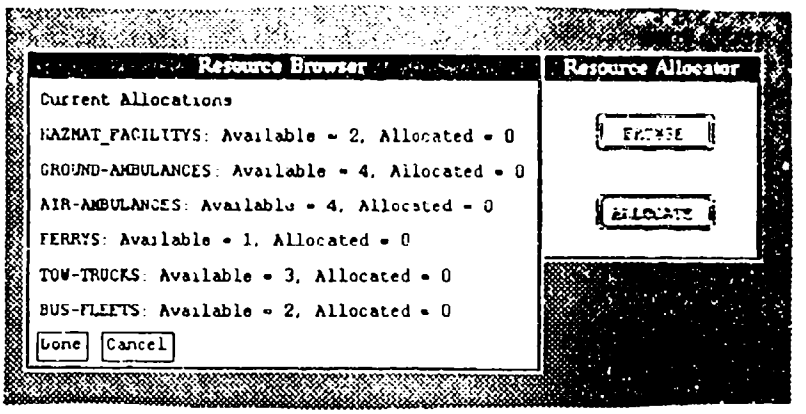

Figure 14: The Resource Allocator.

The central part of this window shows the main information, which could take a variety of forms, including bargraphs, tables, piecharts, and natural language text. Above this information are buttons which allow the user to view previous and succeeding response windows or to redisplay the information in a different way (e.g., barchart changed to a table.) Below the response window is a new editing window for asking followup questions. In the followup section, users can select contexts referring to subsets of the information produced in the window and then ask general questions whose answers will be limited to that subset. For example, after asking for the bedcount of all hospitals in San Mateo County, a user could followup that question by asking for the address of each hospital, and the system will produce the addresses of just those hospitals in San Mateo County. The response windows therefore attempt to provide a simple way of navigating through complex information.

12 The Final System

The prototype emergency management system developed illustrates some of the power of the various capabilities of the CHORIS architecture when applied to a command-and-control-style domain, including: support for natural language and direct manipulation; modeling user's high-level tasks to provide timely guidance in managing complex procedures (seen in the use of the Agenda facility); adapting to individual users and their domain roles (as viewed in the differential support of the system for the EOC Commander vs. the ICS Commander); and the power of dynamically tailoring the presentation of system information to the task needs and preferences of the user (as witnessed in the various response windows produced by the interface in response to user queries). It is hoped that this overview of the CHORIS interface as applied to emergency management in particular and command-and-control domains in general makes clear the value of such an approach to intelligent interface design.

References

[Benbasat84] I. Benbasat and Y. Wand. A structured approach to designing human-computer dialogues. *International Journal of Man-Machine Studies*, 21:105–126, 1984.

[Clark86] N. Clark. The language of data: tables and graphs as exposition. In *Proceedings ICRISAT*, 1986.

[Cleveland84] W. S. Cleveland and R. McGill. Graphical perception: theory, experimentation, and application to the development of graphical methods. *Journal of the American Statistical Association*, 79(387):531 - 554, September 1984.

[Cohen89] P. Cohen, J. Sullivan, M. Dalrymple, R. G. Jr., D. Moran, J. Schlossberg, F. Pereira, and S. Tyler. Synergistic use of direct manipulation and natural language. In *Proceedings of CHI'89*, Austin, TX, May 1989.

[DeSanctis84] G. DeSanctis. Computer graphics as decision aids: directions for research. *Decision Sciences*, 15:463 - 487, 1984.

[Hochberg86] J. Hochberg and D. H. Krantz. Perceptual properties of statistical graphs. In *Proceedings of the Section on Statistical Graphics of the American Statistical Association*, 1986. This paper is a sub-paper in the paper: "Three Perspectives on Statistical Graphs: A Basis for Defining Evaluation Criteria" by Nancy Clark.

[Jr88] R. G. Jr., J. Sullivan, and S. Tyler. Multimodal response planning: an adaptive rule based approach. In *CHI 1988 Proceedings*, May 1988.

[Mackinlay86] J. D. Mackinlay. *Automatic Design of Graphical Presentations*. PhD thesis, Stanford University, December 1986. Report number: STAN-CS-86-1138.

[Tukey86] P. A. Tukey. A data analyst's view of statistical plots. In *Proceedings of the Section on Statistical Graphics of the American Statistical Association*, 1986. This paper is a sub-paper in the paper: "Three Perspectives on Statistical Graphs: A Basis for Defining Evaluation Criteria" by Nancy Clark.

[Tyler86] S. W. Tyler. *SAUCI: A Self-Adaptive User-Computer Interface*. PhD thesis, University of Pittsburgh, October 1986.

[Tyler88] S. Tyler. Sauci: a knowledge based interface architecture. In *CHI 1988 Proceedings*, May 1988.